Apple Assembly Line

$1.80

Volume 4 -- Issue 9 June, 1984

In This Issue...

More on ProDOS and Nonstandard Apples

In the March issue we published Bob Stout's note on how to make ProDOS boot in a Franklin computer. The current issue, (No. 9) of Hardcore Computist points out that the address given in that note didn't work for the ProDOS version dated 1-JAN-84. Apparently Bob was referring to an earlier version. The correct address for the NOPs is $265B.

In a similar vein, inside this issue Jan Eugenides points out that ProDOS will also fail in an Apple with a modified Monitor ROM. He then gives a slightly different patch to defeat the check code.

18-Digit Arithmetic, Part 2...............Bob Sander-Cederlof

Feedback on installment one of this series came from as far away as Sweden. Paul Schlyter, with others, pointed out the omission of three very important letters. PRINT (14.9*10) indeed prints 149, as expected. What I meant to say was that PRINT INT(14.9*10) prints 148.

I noticed another error at the top of page 21. The exponent range runs from 10^-63 thru 10^63, not 10^64.

Paul pointed out that my routines did not check for underflow and overflow. I did have such checks in another part of the code, as yet unlisted, but I now agree with him that some checks belong in the routines printed last month.

The subroutine SHIFT.DAC.RIGHT.ONE is called when a carry beyond the most significant bit is detected in DADD, at line 1620. If the exponent is already 10^63, or $7F, this shift right will cause overflow. That means the sum formed by DADD is greater than 10^63, and we need to do either of two things. My usual choice, assuming the routines are being used from Applesoft, is to JMP directly to the Applesoft ROM overflow error routine, at $E8D5. Another option is to set the DAC exponent to $7F, and the mantissa to all 9's. To implement it my way, add these lines:

```
1945       BMI .2

2085 .2    JMP $E8D5
```

Underflow needs to be tested in the NORMALIZE.DAC subroutine. Underlofw happens when the exponent falls below 10^-63. The normal procedure upon underflow is to set the result to zero. Zero values in DP18 are indicated by the exponent being zero, regardless of the mantissa value. Delete lines 2400-2480 and line 2730, and enter the following lines

```
2400         LDY #-1
2410 .1      INY
2420         CPY #10
2430         BCS .7
2440         LDA DAC.HI,Y
2450         BEQ .1

2730 .6      LDA DAC.EXPONENT
2731         BPL .8
2732 .7      LDA #0
2733         STA DAC.EXPONENT
2734         STA DAC.SIGN
2735 .8      RTS
```

All these changes will be installed on Quarterly Disk 15.

This month I want to present several pack and unpack subroutines, and one which rounds the value in DAC according to the value in the extension byte.

S-C Macro Assembler Version 1.0..................................$80
S-C Macro Assembler Version 1.1...............................$92.50
Version 1.1 Update..$12.50
Source Code for Version 1.1 (on two disk sides)...................$100
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility (without source code).....................$20
S-C Cross Reference Utility (with complete source code)...............$50
DISASM Dis-Assembler (RAK-Ware)...................................$30
Source Code for DISASM...............................additional $30

S-C Word Processor (with complete source code)......................$50
Double Precision Floating Point for Applesoft (with source code)......$50
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk...............................$15

(All source code is formatted for S-C Macro Assembler Version 1.1. Other assemblers require some effort to convert file type and edit directives.)

AAL Quarterly Disks.......................................each $15
Each disk contains all the source code from three issues of "Apple Assembly Line", to save you lots of typing and testing time.

QD#1: Oct-Dec 1980	QD#2: Jan-Mar 1981	QD#3: Apr-Jun 1981
QD#4: Jul-Sep 1981	QD#5: Oct-Dec 1981	QD#6: Jan-Mar 1982
QD#7: Apr-Jun 1982	QD#8: Jul-Sep 1982	QD#9: Oct-Dec 1982
QD#10: Jan-Mar 1983	QD#11: Apr-Jun 1983	QD#12: Jul-Sep 1983
QD#13: Oct-Dec 1983	QD#14: Jan-Mar 1984	QD#15: Apr-Jun 1984

AWIIe Toolkit (Don Lancaster, Synergetics)............................$39
Quick-Trace (Anthro-Digital)..........................(reg. $50) $45
Visible Computer: 6502 (Software Masters).............(reg. $50) $45
ES-CAPE: Extended S-C Applesoft Program Editor........................$60
Amper-Magic (Anthro-Digital)..........................(reg. $75) $65
Amper-Magic Volume 2 (Anthro-Digital)................(reg. $35) $30
Routine Machine (Southwestern Data Systems)..........(reg. $64.95) $60
"Bag of Tricks", Worth & Lechner, with diskette.............($39.95) $36
FLASH! Integer BASIC Compiler (Laumer Research)........................$79
Fontrix (Data Transforms)..$75
Aztec C Compiler System (Manx Software).................(reg. $199) $180

Blank Diskettes (Verbatim)............2.50 each, or package of 20 for $45
(Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
or $25 per 100
These are cardboard folders designed to fit into 6"X9" Envelopes.
Envelopes for Diskette Mailers........................... 6 cents each
ZIF Game Socket Extender (Ohm Electronics) $20

Books, Books, Books........................compare our discount prices!
"Apple][Circuit Description", Gayler.................($22.95) $21
"Understanding the Apple II", Sather...................($22.95) $21
"Enhancing Your Apple II, vol. 1", Lancaster...........($15.95) $15
Second edition, with //e information.
"Incredible Secret Money Machine", Lancaster............($7.95) $7
"Beneath Apple DOS", Worth & Lechner...................($19.95) $18
"Assembly Lines: The Book", Roger Wagner...............($19.95) $18
"What's Where in the Apple", Second Edition............($24.95) $23
"What's Where Guide" (updates first edition).............($9.95) $9
"6502 Assembly Language Programming", Leventhal.........($18.95) $18
"6502 Subroutines", Leventhal..........................($17.95) $17
"Real Time Programming -- Neglected Topics", Foster......($9.95) $9

We have small quantities of other great books, call for titles & prices.
Add $1.50 per book for US shipping. Foreign orders add postage needed.

*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
*** (214) 324-2050 ***
*** We accept Master Card, VISA and American Express ***

Note that I have just LISTed the subroutines below, rather than showing the assembly listing, because the program parts need to all be assembled together before they are meaningful.

There are two "unpack" subroutines, MOVE.YA.DAC and MOVE.YA.ARG. They perform the "load accumulator" function. There is one "pack" subroutine, MOVE.DAC.YA, which performs the "store accumulator" function.

The MOVE routines use a page-zero pair at $5E and $5F. Assuming the DP18 package will be called from Applesoft via the &-vector, there will be no page-zero conflicts here.

The subroutines DADD and DSUB from last month, and DMULT and DDIV to come, all expect two arguments in DAC and ARG and leave the result in DAC. Assuming there are two packed DP18 value at VAL.A and VAL.B, and that I want to add them together and store the result in VAL.C, I would do it this way:

```
        LDA #VAL.A
        LDY /VAL.A
        JSR MOVE.YA.DAC
        LDA #VAL.B
        LDY /VAL.B
        JSR MOVE.YA.ARG
        JSR DADD
        LDA #VAL.C
        LDY /VAL.C
        JSR MOVE.DAC.YA
```

Note that MOVE.DAC.YA calls ROUND.DAC before storing the result. ROUND.DAC checks the extension byte. If the extension byte has a value less than $50, no rounding need be done. If it is $50 through $99, the value in DAC should be rounded up. If the higher digits are less than .999999999999999999, then there will be no carry beyond the most significant digit, and no chance for overflow. However, if it is all 9's we will get a final carry and we will need to change the number to 100000000000000000 and add one to the exponent. In tiny precision, this is like rounding .995 up to 1.00. If the exponent was already 10^63, rounding up with a final carry causes overflow, so I jump to the Applesoft error handler.

```
1000 *SAVE S.DP18 PACK & UNPACK
1010 *--------------------------------
1020 *     ADDRESSES INSIDE APPLESOFT
1030 *--------------------------------
1040 AS.OVRFLW .EQ $E8D5    OVERFLOW ERROR
1050 *--------------------------------
1060 *      PAGE ZERO USAGE
1070 *--------------------------------
1080 PNTR   .EQ $5E,5F
1090 *--------------------------------
1100 *      MOVE (Y,A) INTO DAC AND UNPACK
1110 *--------------------------------
1120 MOVE.YA.DAC
1130        STA PNTR
1140        STY PNTR+1
1150        LDY #9        MOVE 10 BYTES
1160 .1     LDA (PNTR),Y
1170        STA DAC,Y
1180        DEY
1190        BPL .1
1200        INY           Y=0
1210        STY DAC.EXTENSION
```

```
1220        LDA DAC.EXPONENT
1230        STA DAC.SIGN
1240        AND #$7F
1250        STA DAC.EXPONENT
1260        RTS
1270 *--------------------------------
1280 *      MOVE (Y,A) INTO ARG AND UNPACK
1290 *--------------------------------
1300 MOVE.YA.ARG
1310        STA PNTR
1320        STY PNTR+1
1330        LDY #9        MOVE 10 BYTES
1340 .1     LDA (PNTR),Y
1350        STA ARG,Y
1360        DEY
1370        BPL .1
1380        INY           Y=0
1390        STY ARG.EXTENSION
1400        LDA ARG.EXPONENT
1410        STA ARG.SIGN
1420        AND #$7F
1430        STA ARG.EXPONENT
1440        RTS
1450 *--------------------------------
1460 *      PACK AND MOVE DAC TO (Y,A)
1470 *--------------------------------
1480 MOVE.DAC.YA
1490        STA PNTR
1500        STY PNTR+1
1510        JSR ROUND.DAC
1520        LDA DAC.EXPONENT
1530        BIT DAC.SIGN
1540        BPL .1        POSITIVE
1550        ORA #$80      NEGATIVE
1560 .1     LDY #0
1570 .2     STA (PNTR),Y
1580        INY
1590        LDA DAC,Y
1600        CPY #10
1610        BCC .2
1620        RTS
1630 *--------------------------------
1640 *      ROUND DAC BY EXTENSION
1650 *--------------------------------
1660 ROUND.DAC
1670        LDA DAC.EXTENSION
1680        CMP #$50      COMPARE TO .5
1690        BCC .3        NO NEED TO ROUND
1700        LDY #8
1710        SED           DECIMAL MODE
1720 .1     LDA #0
1730        ADC DAC.HI,Y
1740        STA DAC.HI,Y
1750        BCC .2        NO NEED FOR FURTHER PROPAGATION
1760        DEY
1770        BPL .1        ...MORE BYTES
1780        INC DAC.EXPONENT
1790        BMI .4        ...OVERFLOW
1800        LDA #$10      .999...9 ROUNDED TO 1.000...0
1810        STA DAC.HI
1820 .2     CLD
1830 .3     LDA #0
1840        STA DAC.EXTENSION
1850        RTS
1860 .4     CLD
1870        JMP AS.OVRFLW
```

None of the pack/unpack code is especially tricky, but the same cannot be said for DMULT. Multiplication is handled "just like you do it with pencil and paper", but making it happen at all efficiently makes things look very tricky.

Call DMULT after loading the multiplier and multiplicand into DAC and ARG (doesn't matter which is which, because multiplication is commutative). Then JSR DMULT to perform the multiply. The result will be left in DAC.

Looking at the DMULT code, lines 1040-1070 handle the special cases of either argument being 0. Anything times zero is zero, and zero values are indicated by the exponent being zero, so this is real easy.

Lines 1090-1130 clear a temporary register which is 20 bytes long. This register will be used to accumulate the partial products. Just in case some of the terminology is losing you, here are some definitions:

```
      12345  <-- multiplicand
    x 54321  <-- multiplier
  ---------
      12345  <-- 1st partial product
     24690   <-- 2nd partial product
    37035    <-- 3rd    "       "
   49380     <-- 4th    "       "
  61725      <-- 5th    "       "
  ---------
  670592745  <-- product
```

Lines 1150-1180 form the 20-digit product of the two 10-digit arguments. I wanted to reduce the number of times the individual digits have to be isolated, or the accumulators shifted by 4-bits, so I used a trick. Line 1150 calls a subroutine which multiplies the multiplicand (in ARG) by all the low-order digits in each byte of the multiplier (in DAC). In other words, I accumulate only the odd partial products at this time. Then I shift DAC 4-bits right, which places the other set of digits in the low-order side of each byte. I also have to shift the result register, MAC, right 4-bits, and then I call the MULTIPLY.BY.LOW.DIGITS subroutine again.

Lines 1200-1270 form the new exponent, which is the sum of the exponents of the two arguments. Since both exponents have the value $40 added to make them appear positive, one of the $40's has to be subtracted back out. But before that, if the sum is above $C0 then we have an overflow condition. After subtracting out one of the $40's, if the result is negative we have an underflow condition. Note that since the carry status was clear at line 1250, I subtracted $3F; for one more byte, I could have done it the normal way and used SEC, SBC #$40.

Lines 1290-1310 form the sign of the product, which is the exclusive-or of the signs of the two arguments. Lines 1330-1370 copy the most significant 10 bytes of the product from MAC to DAC.

The result may have a leading zero digit in the left half of the first byte, so I call NORMALIZE.DAC at line 1390. If The leading digit was zero, normalizing will shift DAC left one digit position, leaving room for another significant digit on the right end. Lines 1400-1490 handle installing the extra digit if necessary.

MULTIPLY.BY.LOW.DIGITS picks up the low-order digit out of each byte of the multiplier, one-by-one, and calls MULTIPLY.ARG.BY.N.

MULTIPLY.ARG.BY.N does the nitty-gritty multiplication. And here is where I lost all my ingenuity, too. The multiplier digit is stored in DIGIT, and used to count down a loop which adds ARG to MAC DIGIT times. Surely this can be done more efficiently! How about it Paul? Or Charlie? Anyone?

```
1000 *SAVE S.DP18 MULTIPLY
1010 *--------------------------------
1020 * DAC = ARG * DAC
1030 *--------------------------------
1040 DMULT  LDA DAC.EXPONENT  IF DAC=0, EXIT
1050        BEQ .3
1060        LDA ARG.EXPONENT  IF ARG=0, SET DAC=0 AND EXIT
1070        BEQ .4
1080 *---CLEAR RESULT REGISTER--------
1090        LDA #0
1100        LDY #19
1110 .1     STA MAC,Y
1120        DEY
1130        BPL .1
1140 *---FORM PRODUCT OF FRACTIONS----
1150        JSR MULTIPLY.BY.LOW.DIGITS
1160        JSR SHIFT.MAC.RIGHT.ONE
1170        JSR SHIFT.DAC.RIGHT.ONE
1180        JSR MULTIPLY.BY.LOW.DIGITS
1190 *---ADD THE EXPONENTS------------
1200        LDA DAC.EXPONENT
1210        CLC
1220        ADC ARG.EXPONENT
1230        CMP #$C0      CHECK FOR OVERFLOW
1240        BCS .5        ...OVERFLOW
1250        SBC #$3F      ADJUST OFFSET
1260        BMI .4        ...UNDERFLOW
1270        STA DAC.EXPONENT
1280 *---FORM SIGN OF PRODUCT---------
1290        LDA DAC.SIGN
1300        EOR ARG.SIGN
1310        STA DAC.SIGN
1320 *---MOVE MAC TO DAC--------------
1330        LDY #9
1340 .2     LDA MAC,Y
1350        STA DAC.HI,Y
1360        DEY
1370        BPL .2
1380 *---NORMALIZE DAC----------------
1390        JSR NORMALIZE.DAC
1400        LDA MAC       IF LEADING DIGIT=0,
1410        AND #$F0      THEN GET ANOTHER DIGIT
1420        BNE .3
1430        LDA MAC+10
1440        LSR
1450        LSR
1460        LSR
1470        LSR
1480        ORA DAC.HI+9
1490        STA DAC.HI+9
1500 .3     RTS
1510 .4     LDA #0
1520        STA DAC.SIGN
1530        STA DAC.EXPONENT
1540        RTS
1550 .5     JMP AS.OVRFLW
1560 *--------------------------------
1570 *      MULTIPLY BY EVERY OTHER DIGIT
1580 *--------------------------------
1590 MULTIPLY.BY.LOW.DIGITS
1600        SED           DECIMAL MODE
1610        LDX #9
1620        LDY #19
1630 .1     LDA DAC.HI,X
1640        AND #$0F      ISOLATE NYBBLE
1650        BEQ .2        0, SO NEXT DIGIT
1660        JSR MULTIPLY.ARG.BY.N
1670 .2     DEY           NEXT MAC POSITION
1680        DEX           NEXT DAC DIGIT
1690        BPL .1        DO NEXT DIGIT
```

```
1700          CLD            BINARY MODE
1710          RTS            DONE
1720 *--------------------------------
1730 MULTIPLY.ARG.BY.N
1740          STA DIGIT      N = 1...9
1750          STY TEMP       SAVE Y
1760          STX TEMP+1     SAVE X
1770 .1       LDX #9         INDEX INTO ARG
1780          CLC
1790 .2       LDA ARG.HI,X
1800          ADC MAC,Y      ADD IT
1810          STA MAC,Y
1820          DEY            NEXT MAC
1830          DEX            NEXT ARG
1840          BPL .2         NEXT DIGIT
1850          BCC .4         NO CARRY
1860 .3       LDA #0         PROPAGATE CARRY
1870          ADC MAC,Y
1880          STA MAC,Y
1890          DEY
1900          BCS .3         MORE CARRY
1910 .4       LDY TEMP       GET POSITION IN MAC
1920 .5       DEC DIGIT      NEXT DIGIT
1930          BNE .1
1940          LDX TEMP+1
1950          RTS            DONE
1960 *--------------------------------
1970 SHIFT.MAC.RIGHT.ONE
1980          LDY #4         4 BITS RIGHT
1990 .0       LDX #1         20 BYTES
2000          LSR MAC
2010 .1       ROR MAC,X
2020          INX            NEXT BYTE
2030          PHP
2040          CPX #20
2050          BCS .2         NO MORE BYTES
2060          PLP
2070          JMP .1
2080 .2       PLP
2090          DEY            NEXT BIT
2100          BNE .0
2110          RTS
2120 *--------------------------------
```

Well, that's all for this month. Next month expect some simple I/O routines and the divide subroutine.

DOSology and DOSonomy..........................Bob Sander-Cederlof

The other day I was thinking about the way Apple spells ProDOS. They jealously guard the spelling, having trademarked the idea of upper-case "P" and "DOS" with lower-case "ro".

Of course, we all know that "Pro" is a standard prefix, with origins in the Greek language. In Greek it means "before". I think Apple derived it from the English word "professional", so that ProDOS means "professional DOS". Nevertheless, the "pro" even in the word professional means before, according to the etymologies in dictionaries.

I took some Greek courses at Dallas Theological Seminary back in 1973 and 1974. I remember very little now, but one thing stuck with me: prepositions. "Pro" is one, but there are a lot more. What other interesting DOSses can we invent?

By the way, the preferred pronunciation of DOS rhymes with "boss", not "gross". If you insist on rhyming with the latter, your pronunciation has a decided Spanish accent. For you we have invented "UnoDOS", which is of course two-thirds of a popular product on the IBM-PC, uno-dos-tres by Lotus. Ha!

The first that came to mind was "ParaDOS". We like it so well, we'd like to trademark it! It could relate to either paradox or pair-of-dice or paradise, take your pick. A shrewdly written DOS could appear as all three at different times to different people.

Bill and I then started to brainstorm, and we can't stop. We've got a blackboard full of neat names, just waiting for some one to write code for. We may have stumbled on to some previously-used names, like SoliDOS and ProntoDOS, but for the most part I think we have cornered the market.

AmbiDOS	MisoDOS	PhiloDOS	BiblioDOS	ViviDOS	DiaDOS
PaleoDOS	MesoDOS	NeoDOS	PsychoDOS	MoriDOS	Dial-a-DOS
ChromoDOS	BlancoDOS	TechniDOS	SomatoDOS	DulciDOS	AnoDOS
AcriDOS	FeloniDOS	BaloniDOS	FormiDOS	MiniDOS	CathoDOS
MicroDOS	MidiDOS	MilliDOS	MegaDOS	NanoDOS	VagaDOS
TeraDOS	UniDOS	BioDOS	StupiDOS	TorriDOS	FabriDOS
SemiDOS	PeriDOS	AntiDOS	AnteDOS	ProsDOS	ExoDOS
HypoDOS	HyperDOS	OvaDOS	PupaDOS	PropoDOS	EnDOS
ArcheDOS	StatiDOS	DynamoDOS	DynaDOS	ProtoDOS	EschatoDOS
OsteoDOS	MultiDOS	PuroDOS	CardioDOS	PyroDOS	PrimaDOS
FrigiDOS	InterDOS	AndroDOS	GynoDOS	GymnoDOS	PseudoDOS
HieroDOS	SpiroDOS	HelioDOS	CycloDOS	AutoDOS	AggreDOS
ManoDOS	ChiroDOS	PetroDOS	LithoDOS	AeroDOS	PosiDOS
PlanoDOS	LiquiDOS	MarbleDOS	PedoDOS	GraviDOS	NegaDOS
PedaDOS	GeriaDOS	NutriDOS	FlexiDOS	PleniDOS	NecroDOS
VisiDOS	InvisiDOS	FluoriDOS	FloriDOS	FaunaDOS	PensaDOS
ThanaDOS	AgriDOS	NaviDOS	NovaDOS	SpuriDOS	MensaDOS
StereoDOS	VerbiDOS	VermiDOS	CineDOS	GeoDOS	TragiDOS
MonoDOS	DuoDOS	CobraDOS	FerroDOS	OxyDOS	AfroDOS
EuroDOS	NippoDOS	FrancoDOS	IndoDOS	CanaDOS	HispanoDOS

Get the idea?

OBJ.APWRT][F updated to AWIIe Toolkit..........Don Lancaster

I have packed even more goodies on eight disk sides, combining the HACKER and USER packages into one powerful Toolkit. The price is only slightly higher... They were $29.50 each, now only $39.50 together.

Now that we have yet another Apple monitor, vastly different yet purportedly compatible, guess what! Applewriter //e is not QUITE compatible with the //c. Surprise, surprise! The status line display gets turned into garbage. One of the patches included in the new AWIIe Toolkit solves the problem admirably. This AWIIE CLARIFIER Applesloth program modifies your Applewriter IIe backup diskettes to eliminate trashing of the IIc status display line. Here it is now, more than slightly compressed for AAL, to tease you into getting the whole Toolkit:

```
100  REM *--------------------------------------*
200  REM *  COPYRIGHT 1984 BY DON LANCASTER AND  *
220  REM *  SYNERGETICS,  BOX 1300, THATCHER AZ  *
240  REM *  85552       Phone:  (602) 428-4073   *
260  REM *   ALL  COMMERCIAL RIGHTS RESERVED     *
280  REM *--------------------------------------*
380  TEXT : HOME : HIMEM: 8000
400  HTAB 8: PRINT "Applewriter IIe CLARifier": PRINT
600  REM Check Validity
660  PRINT CHR$(4)"BLOAD OBJ.APWRT][F,A$2300
670  IF  PEEK (14815) <  > 188 THEN 880
680  IF  PEEK (15052) <  > 41 THEN 880
690  IF  PEEK (15096) <  > 59 THEN 880
695  REM Install Patches
700  POKE 14815,60: POKE 14816,36: POKE 14817,207:
      POKE 14818,16: POKE 14819,2: POKE 14820,169:
      POKE 14821,62
710  POKE 15052,208: POKE 15053,42
720  POKE 15062,96
730  POKE 15096,41: POKE 15097,127: POKE 15098,201:
      POKE 15099,96: POKE 15100,176: POKE 15101,208:
      POKE 15102,201: POKE 15103,64
740  POKE 15104,144: POKE 15105,204: POKE 15106,41:
      POKE 15107,63: POKE 15108,176: POKE 15109,200
750  PRINT CHR$(4)"UNLOCK OBJ.APWRT][F"
760  PRINT CHR$(4)"BSAVE OBJ.APWRT][F,A$2300,L$30D3"
770  PRINT CHR$(4)"LOCK OBJ.APWRT][F"
870  PRINT "IT WORKED!" : END
880  PRINT "Will not verify as AWIIe; patch ABORTED" : END
```

Gotchas: Fixes only the status line. Rare and brief changes in the flashing cursor symbol will remain.

Using the PRT Command.................................Bill Morgan

New users of the S-C Macro Assembler have asked for examples of how to use some of the customizing features. For example, just now I had a call from a gentleman who needed to know how to set up the PRT vector to turn on his printer and send the special control strings it requires.

It happens that I had the same problem just a few weeks ago. I just picked up an OkiData 92 printer, which I am quite happy with, except for a couple of small warts. Setting Elite spacing (12 characters/inch, 8 lines/inch) on that printer requires these hex codes: 9C 9B B8. The catch is that 9C, which corresponds to Control-backslash. I can't type CTRL-\ on my Apple II+! Besides, by the time I type in the commands to turn on the printer, set Elite mode, and set a left margin, I have entered 15 keystrokes. That's too many for my lazy, dyslexic fingers, so I came up with a PRT command to do the whole job.

The addresses in this routine are set up for the 40-column Version 1.1 Language Card assembler. If you are using another version, check to make sure that the patch space is indeed all zeroes. All $D000 versions of the assembler have some blank space before $E000. If you are using a $1000 version, look to see if there is some space available between the end of the assembler and the beginning of the Symbol Table and set PATCH.SPACE to that address. You will also have to set PRT.VECTOR to $1009.

Here are the exact steps to use this patch:

```
Start the assembler.

$C083 C083
$D01C:0 D0 0 F8

$AA60.AA61

LOAD S.PRT

ASM

$D01C:0 0 0 0
$C080

BSAVE <assembler>,A$D000,L$XXXX
```

The $AA60.AA61 line gives you the length that you will need to use for the BSAVE command. Substitute the filename of the version you use for <assembler> in the above command.

If you are using Version 1.0 of the assembler, things are a little different. You should omit the $D01C entries in the above commands, delete lines 1090 and 1100, and add this line to the program:

```
1125       .TA $800
```

Then, after the assembly, install the patch with $DF00<800.81EM and $D009: 4C 00 DF. These extra steps are necessary because Version 1.0 lacks the ability to override memory protection during assembly.

Lines 1270-1290 are where you should install the codes your printer needs.

```
                  1000 *--------------------------------
                  1010 *
                  1020 *   SAMPLE PRT COMMAND
                  1030 *
                  1040 *--------------------------------
D009-             1050 PRT.VECTOR  .EQ $D009
DF00-             1060 PATCH.SPACE .EQ $DF00
FDED-             1070 MON.COUT    .EQ $FDED
                  1080 *--------------------------------
                  1090        .OR PRT.VECTOR
D009- 4C 00 DF    1100        JMP PRT           JUMP TO HANDLER
                  1110 *--------------------------------
                  1120        .OR PATCH.SPACE
                  1130
DF00- A2 00       1140 PRT    LDX #0
DF02- BD 0E DF    1150 .1     LDA STRING,X      OUTPUT THE
DF05- F0 06       1160        BEQ .2            CONTROL
DF07- 20 ED FD    1170        JSR MON.COUT      STRING
DF0A- E8          1180        INX
DF0B- 10 F5       1190        BPL .1
                  1200
DF0D- 60          1210 .2     RTS
                  1220 *--------------------------------
DF0E- 8D 84       1230 STRING .HS 8D84          <CR><^D>
DF10- D0 D2 A3
DF13- B1          1240        .AS -/PR#1/
DF14- 8D          1250        .HS 8D            <CR>
                  1260
DF15- 9C 9B B8    1270        .HS 9C9BB8        ELITE SPACING
DF18- 9B A5 C3    1280        .HS 9BA5C3        LEFT MARGIN
DF1B- B0 B9 B0    1290        .HS B0B9B0        90 DOT SPACES
DF1E- 00          1300        .HS 00            END MARKER
```

Revisiting $48:0............................Bob Sander-Cederlof

Remember all those warnings about storing 0 in $48 after DOS had a whack at your zero page? Maybe not, but let me remind you.

Apple's monitor uses locations $45 through $49 in a very special way. Ignoring this, the writers of DOS also used them. When you start execution from the monitor (using the G, S, or T commands) The data in these locations gets loaded into the registers: $45 into A, $46 into X, $47 into Y, $48 into P (status), and $49 into S (stack pointer). When a program hits a BRK opcode, or the S command has finished executing a single opcode, the monitor saves these five registers back into $45...$49.

No serious problem, unless you like to enter the monitor and issue the G, S, or T commands. Even less of a problem, because the S and T commands were removed from the monitor ROM when the Apple II Plus came out. And if you don't care what is in the registers anyway....

But the P-register is rather special, too. One of its bits, called "D", controls how arithmetic is performed. If "D" is zero, arithmetic will be done in the normal binary way; if D=1, arithmetic is done in BCD mode. That is, adding one to $49

will produce $50 rather than $4A. If the program you are entering doesn't expect to be in decimal mode, and tries arithmetic, you will get some rather amusing results.

Hence the warning: before using the G command from the monitor, type 48:0 to be sure decimal mode is off. Later versions of DOS store 0 into $48 after calling those routines which use $48. And the monitor stores 0 into $48 whenever you hit the RESET key (or Control-RESET).

```
*****************************************************
*                                                   *
*   Now I am here to tell you that storing 0 into   *
*   $48 is ALL WRONG! It took Bill and me 5 hours   *
*   to unravel the mystery caused by storing zero   *
*   there!                                          *
*                                                   *
*****************************************************
```

You should put into $48 a sensible value. Better, DOS should never use $45 through $48; if it must use them, save and restore them. There are eight bits in the P-register, and in the 6502 seven of them are important. One of them, we discovered, is VERY important.

The bit named "I" controls the IRQ interrupt. If I=1, IRQ interrupts will not be accepted. If I=0, IRQ interrupts will be accepted. So...who cares about interrupts?

Hardly anyone uses interrupts in Apple II's, because of all the hidden problems. But there are some very nice boards for the Apple that are designed to be used with interrupts. Most of them are safe, because RESET disables their interrupt generators.

Need I say that we discovered a board that does not disable the interrupt generators when you hit RESET? The Novation Cat Modem (a very excellent product) leaves at least one of its potential IRQ sources in an indeterminate state. IRQ's don't immediately show up, though, because they are trapped until you have addressed any of the soft switches on the card. But, for example, if that card is in slot 2 and I read or write any location from $C0A0 through $C0AF, IRQ's start coming. Still no problem, because I=1 in the P-register.

UNTIL WE USE THE MONITOR G COMMAND!

If I use the monitor G command, location $48, containing 0, is loaded into the P-register. Then an IRQ gets through and sends the 6502 vectoring through an unprepared vector at $3FE,3FF and BANG!

Our solution was to put SEI instructions in various routines, and to make sure that $48 contains 4, not 0, before using the G command.

From now on, whenever you hear that you need to be sure $48 contains zero, think four.

More Random Number Generators..............Bob Sander-Cederlof

I published my "Random Numbers for Applesoft" article last month just in time. The June issue of Micro includes a 9.5 page article called "A Better Random Number Generator", by H. Cem Kaner and John R. Vokey. The article reports on work funded by the Natural Sciences and Engineering Research Council of Canada (NSERC).

The authors give an excellent overview of the various methods used to test random number generators, and the methods they used during their seven years of research to produce three "better" generators. It is worth buying a copy of Micro to get a copy of this article.

The authors used the same linear congruential algorithm I discussed last month, but with different parameters. My favorite last month was:

```
        R(new) = ( R(old) * A + C ) mod 2^32
        where A = 314159269
          and C = 907633386
```

Kaner and Vokey decided to use a 40-bit seed and use mod 2^40 in calculating each successive value. After extensive analysis and testing, they came up with three generators based on the following values for "A" and "C":

```
        RNG 1:  A = 31415938565
                C = 26407

        RNG 2:  A = 8413453205
                C = 99991

        RNG 3:  A = 27182819621
                C = 3
```

There are an unusually large number of typos in the article, and some of them are hard to decipher. The value 26407 above was written in the comment field as 24607; however, in the hexadecimal constant assembly code it was 0000006727, which is 26407. Even worse, in the listing there are missing lines of code and missing characters here and there. All of the immediate mode instructions are missing a "#" character. Four or five labels are never defined in the listing.

Since the program as printed cannot possibly be successfully loaded, assembled, POKEd, or executed, I have chosen to re-write it for inclusion here, after my own style. I believe my version is also significantly improved in coding and speed.

The reason given for choosing to work with 40 bits rather than 32, even though Applesoft only stores 32 and using 40 takes longer, was that it is important to provide more values between 0.0 and 2^-32. I tend to disagree on the importance of this, since most uses of random numbers on the Apple are for games, and simulate such simple things as dealing cards or throwing dice. Perhaps more serious simulations need more precise

randomness. Of course they also increase by a factor of 256 the number of numbers generated before the sequence repeats.

Buried in the middle of the program is a well-optimized 40-bit multiplication loop. You might enjoy puzzling out how it works!

The program uses USR(x), where x selects which of the three generators to use. There is no provision for setting the seed or for selecting a range other than 0...1, such as I included in my programs last month. Some enterprising individual will marry the shell of my USR subroutine with the RNG of Kaner and Vokey to produce a really great Applesoft Random Number Generator.

```
                  1000 *SAVE S.KANER & VOKEY
                  1010 *--------------------------------
                  1020 *      BASED ON PROGRAM PRINTED IN MICRO MAGAZINE
                  1030 *      JUNE 1984, PAGES 33,34, BY H. CEM KANER
                  1040 *                                AND JOHN R. VOKEY
                  1050 *--------------------------------
000A-             1060 USER.VECTOR .EQ $0A,0B,0C
009D-             1070 FAC         .EQ $9D THRU $A1
00A2-             1080 FAC.SIGN    .EQ $A2
00AC-             1090 FAC.EXT     .EQ $AC
                  1100 *--------------------------------
E82E-             1110 NORMALIZE.FAC.2 .EQ $E82E
                  1120 *--------------------------------
                  1130        .OR $300
                  1140        .TF B.KANER & VOKEY
                  1150 *--------------------------------
0300- A9 4C       1160 LINK   LDA #$4C      "JMP" OPCODE
0302- 85 0A       1170        STA USER.VECTOR
0304- A9 45       1180        LDA #RANDOM
0306- 85 0B       1190        STA USER.VECTOR+1
0308- A9 03       1200        LDA /RANDOM
030A- 85 0C       1210        STA USER.VECTOR+2
030C- 60          1220        RTS
                  1230 *--------------------------------
030D- 00 00 00
0310- 67 27       1240 Z.C    .HS 00.00.00.67.27  26407
0312- 07 50 89
0315- 2E 05       1250 Z.A    .HS 07.50.89.2E.05  31415938565
0317- 00 00 00
031A- 00 00       1260 Z.OLD  .HS 00.00.00.00.00
                  1270 *--------------------------------
031C- 00 00 01
031F- 86 97       1280 Y.C    .HS 00.00.01.86.97  99991
0321- 01 F5 7B
0324- 1B 95       1290 Y.A    .HS 01.F5.7B.1B.95  8413453205
0326- 00 00 00
0329- 00 00       1300 Y.OLD  .HS 00.00.00.00.00
                  1310 *--------------------------------
032B- 00 00 00
032E- 00 03       1320 X.C    .HS 00.00.00.00.03  3
0330- 06 54 38
0333- E9 25       1330 X.A    .HS 06.54.38.E9.25  27182819621
0335- 00 00 00
0338- 00 00       1340 X.OLD  .HS 00.00.00.00.00
                  1350 *--------------------------------
033A-             1360 GROUP      .BS 1
033B-             1370 MULTIPLIER .BS 5
0340-             1380 OLD        .BS 5
                  1390 *--------------------------------
0345- A0 04       1400 RANDOM LDY #Z.C-Z.C+4
0347- A5 A2       1410        LDA FAC.SIGN
0349- 30 08       1420        BMI .1        SELECT Z
034B- A0 13       1430        LDY #Y.C-Z.C+4
034D- A5 9D       1440        LDA FAC
034F- F0 02       1450        BEQ .1        SELECT Y
0351- A0 22       1460        LDY #X.C-Z.C+4   SELECT X
0353- 8C 3A 03    1470 .1     STY GROUP     SAVE FOR LATER
```

```
                       *---LOAD SELECTED GROUP----------
0356- A2 04                    LDX #4          MOVE 5 BYTES
0358- B9 0D 03         .2      LDA Z.C,Y
035B- 95 9E                    STA FAC+1,X
035D- B9 12 03                 LDA Z.A,Y
0360- 9D 3B 03                 STA MULTIPLIER,X
0363- B9 17 03                 LDA Z.OLD,Y
0366- 9D 40 03                 STA OLD,X
0369- 88                       DEY
036A- CA                       DEX
036B- 10 EB                    BPL .2
                       *---MULTIPLY INTO FAC------------
036D- A2 04                    LDX #4
036F- 86 AC            .3      STX FAC.EXT     USE FOR BYTE COUNT
0371- BD 3B 03                 LDA MULTIPLIER,X
0374- 85 9D                    STA FAC         SAVE FOR 8-BIT MULITPLY
0376- A0 07                    LDY #7          COUNT BITS
0378- 46 9D            .4      LSR FAC         GET RIGHTMOST BIT INTO CARRY
037A- 90 0B                    BCC .6          =0, SO WE DO NOT ADD THIS TIME
037C- 18                       CLC             =1, SO WE BETTER ADD
037D- B5 9E            .5      LDA FAC+1,X
037F- 7D 40 03                 ADC OLD,X
0382- 95 9E                    STA FAC+1,X
0384- CA                       DEX
0385- 10 F6                    BPL .5
0387- 0E 44 03         .6      ASL OLD+4       SHIFT MULTIPLICAND LEFT
038A- 2E 43 03                 ROL OLD+3
038D- 2E 42 03                 ROL OLD+2
0390- 2E 41 03                 ROL OLD+1
0393- 2E 40 03                 ROL OLD
0396- A6 AC                    LDX FAC.EXT     GET BYTE COUNT AGAIN
0398- 88                       DEY             NEXT BIT
0399- 10 DD                    BPL .4
039B- CA                       DEX             REDUCE BYTE COUNT
039C- 10 D1                    BPL .3          ...MORE BYTES
                       *---SAVE NEW SEED IN OLD---------
039E- A2 04                    LDX #4
03A0- AC 3A 03                 LDY GROUP
03A3- B5 9E            .7      LDA FAC+1,X
03A5- 99 17 03                 STA Z.OLD,Y
03A8- 88                       DEY
03A9- CA                       DEX
03AA- 10 F7                    BPL .7
                       *---NORMALIZE NEW VALUE----------
03AC- A0 80                    LDY #$80        ASSUME A FRACTION
03AE- A5 9E            .8      LDA FAC+1       LOOK AT LEADING BIT
03B0- 30 11                    BMI .9          ...FINISHED NORMALIZING
03B2- 46 A2                    LSR FAC.SIGN
03B4- 66 A1                    ROR FAC+4
03B6- 66 A0                    ROR FAC+3
03B8- 66 9F                    ROR FAC+2
03BA- 66 9E                    ROR FAC+1
03BC- 88                       DEY
03BD- C0 58                    CPY #$58
03BF- B0 ED                    BCS .8
03C1- A0 00                    LDY #0          LESS THAN 2^-40 IS ZERO
03C3- 84 9D            .9      STY FAC         EXPONENT
03C5- A9 00                    LDA #0
03C7- 85 A2                    STA FAC.SIGN MAKE IT POSITIVE
03C9- 60                       RTS
```

Booting ProDOS with a Modified Monitor ROM........Jan Eugenides

You may have already figured this out, but ProDOS won't boot if you have installed S. Knouse's modified ROM in your Apple. This can easily be fixed, as follows:

- On track 1, sector C, change bytes B4-B6 from AE B3 FB to A2 EA EA. This tells ProDOS your machine is a II+. If it's a //e, make B5 an A0 instead.
- On track 1, sector 9, change bytes 60-61 from A9 00 to A5 0C. This defeats the ROM check routine.

Ta daaa! Now ProDOS works just fine with your modified ROM.

Fixing the Andromeda 16K Card.....................Bob Bernard

In the April 1984 Call-APPLE there was a letter from John Wallace regarding a problem with the Andromeda 16K RAM card. As this card was the second on the market, first after Apple's (which was bundled with Pascal), there are probably still tens of thousands in use. Yet the Andromeda is anathema to some hardware and software.

In particular, it played havoc with John Wallace's copy of Apple PIE (a popular word processor from yesteryear), and my Lobo 8" floppy drive controller (another relic, I suppose). Bob S-C tells of running into the problem too:

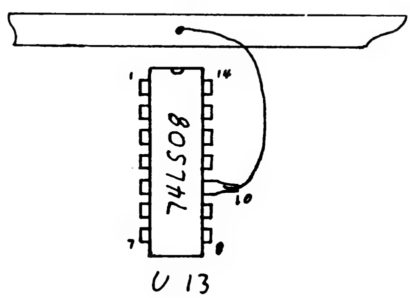
> "I have an Andromeda board, and I ran into this problem with early versions of ES-CAPE. Using a STA (or other store) opcode to any soft switches on the Andromeda card write-protected the card. Using two stores in a row to try to write-enable the card does no good either. I had to change all stores to loads or BITs to make it work. Apple's board accepts either stores or loads, as do all other memory cards I have tested."

There are probably lots of interfaces and programs out there which stumble over Andromeda. Wallace details a hardware modification to the Andromeda board which makes it work the same as all other memory boards. I found a slightly simpler way, and I recommend that all Andromeda owners fix their boards as soon as possible.

Remove the 74LS08 chip at board location U13. Bend pin 10 out so that it sticks straight out, and plug the chip back into its socket so that pin 10 is on the outside. Solder a small wire to pin 10 (carefully), and solder the other end of the wire to pin 14 of the same chip. Or, you can solder to a solder pad pin 14 is connected to, as shown in the drawing below. (Pin 14 is connected to Vcc, the +5 volts line.) That's all there is to it.

John Wallace suggests using a 1K resistor rather than a wire, but I found a wire is sufficient.

With the wire installed, both reads and writes can be used to switch the card, just like Apple intended it.



Finding the Erroneous Bit Using CRC................Bruce Love
Hamilton, New Zealand

The April 1984 AAL article about using Cyclic Redundancy Codes posed the question, "How do you find out which bit was in error, assuming only one was wrong?" I found a way.

My algorithm assumes that there was one and only one bit wrong in the entire 258-byte message (256 bytes of original message plus 2 bytes of CRC). The bits are numbered left-to-right, or most significant bit of first byte received through the least significant bit of the CRC, 0 through $80F (or 2063, if you prefer decimal).

After receiving the data and CRC, the RECV program has computed a composite CRC and the result will be $0000 if there were no errors. If the result is non-zero, it uniquely determines which bit was wrong. Here is a summary of my algorithm for finding which bit:

```
    let bit.number = 2063
    let dummy.crc  = 1
--->if dummy.crc = crc, then we found the bit
|   decrement bit.number
|   shift dummy.crc left 1 bit
|   if carry set, EOR with $1021
----loop
```

[The following comments are by Bob Sander-Cederlof.]

The program listing which follows is an addendum to the listing in the April issue of AAL. Lines 3070 through the end should be appended to the program in that issue. If you buy the AAL Quarterly Disk, it will already be there.

The sequence I used for testing the program went like this. First I assembled the whole program, April's plus the one below. Then I typed "$4000<F800.F8FFM" to move a copy of the monitor's first page into the test buffer. I thought this would be "interesting" data to play with. Then these steps:

```
:MGO SEND          (fakes sending the buffer)
1F45               (SEND prints out the CRC for BUFFER)
:$4000             (see what is there)
4A                 (it was $4A)
:$4000:CA          (make a fake error in the 1st bit)
:MGO RECV
xxxx               (some non-zero value)
:MGO FIND.BAD.BIT
0000               (the bad bit was the first bit)
$4000:4A           (restore the correct bit
```

Then I tried the same steps on various other bit positions, with accurate results in every case.

```
                    3060 *--------------------------------
                    3070 *   FIND WHICH BIT IS BAD IN BUFFER+CRC
                    3080 *
                    3090 *       RESULT IS BIT POSITION IN MESSAGE,
                    3100 *       WHERE THE FIRST BIT OF THE MESSAGE IS BIT 0
                    3110 *       AND (IN THIS CASE) THE LAST CRC BIT IS BIT $80F.
                    3120 *
                    3130 *       ALGORITHM BY BRUCE LOVE, NEW ZEALAND
                    3140 *--------------------------------
0010-               3150 BIT.NUMBER .EQ $10,11
0012-               3160 DUMMY.CRC  .EQ $12,13
                    3170 *--------------------------------
                    3180 FIND.BAD.BIT
0954- A9 0F         3190        LDA #$80F    TOTAL # BITS - 1
0956- 85 10         3200        STA BIT.NUMBER    (WE WILL COUNT BACKWARDS)
0958- A9 08         3210        LDA /$80F
095A- 85 11         3220        STA BIT.NUMBER+1
095C- A9 01         3230        LDA #$0001   STARTING POINT FOR BIT FINDER
095E- 85 12         3240        STA DUMMY.CRC
0960- A9 00         3250        LDA /$0001
0962- 85 13         3260        STA DUMMY.CRC+1
0964- A5 00         3270 .1     LDA CRC      COMPARE RECEIVED CRC WITH
0966- C5 12         3280        CMP DUMMY.CRC         PROCESSED VALUE;
0968- D0 06         3290        BNE .2       IF THEY MATCH, WE HAVE FOUND THE
096A- A5 01         3300        LDA CRC+1    BAD BIT.
096C- C5 13         3310        CMP DUMMY.CRC+1
096E- F0 1F         3320        BEQ .4       ...FOUND BAD BIT!
0970- A5 10         3330 .2     LDA BIT.NUMBER        DECREMENT BIT COUNTER
0972- D0 04         3340        BNE .3
0974- C6 11         3350        DEC BIT.NUMBER+1
0976- 30 24         3360        BMI .5       WENT TOO FAR
0978- C6 10         3370 .3     DEC BIT.NUMBER
097A- 06 12         3380        ASL DUMMY.CRC
097C- 26 13         3390        ROL DUMMY.CRC+1
097E- 90 E4         3400        BCC .1
0980- A5 12         3410        LDA DUMMY.CRC
0982- 49 21         3420        EOR #$21
0984- 85 12         3430        STA DUMMY.CRC
0986- A5 13         3440        LDA DUMMY.CRC+1
0988- 49 10         3450        EOR #$10
098A- 85 13         3460        STA DUMMY.CRC+1
098C- 4C 64 09      3470        JMP .1
098F- A5 11         3480 .4     LDA BIT.NUMBER+1  PRINT THE BIT NUMBER
0991- 20 DA FD      3490        JSR PRBYTE        (IF $8000, THE ERROR WAS
0994- A5 10         3500        LDA BIT.NUMBER    NOT A SINGLE BIT)
0996- 20 DA FD      3510        JSR PRBYTE
0999- 4C 8E FD      3520        JMP CROUT
099C- 00            3530 .5     BRK
                    3540 *--------------------------------
```

The Barkovitch Utilities

Did you notice Dave Barkovitch's ad last month? He has written a very handy set of utilities for us serious Applers, and sells 'em cheap! Be prepared to puzzle your way through his admittedly skimpy documentation, but it is all there.

The I/O Tracer comes in EPROM on a little card that plugs into any slot 1-7 for only $40.50 (including shipping). I/O Tracer is essentially a powerful disk ZAP utility, allowing you to read/write/edit any DOS 3.3 sector. You see an entire sector at once on the screen, in either hex or ASCII, along with all status information.

Dave's Single-Step Trace program will help you debug assembly language. He likes it better than the other commercial varieties of debuggers, and sells it for only $35.

Any questions, call Dave at (201) 499-0636.

Converting to Motorola S-Format...........Bob Sander-Cederlof

Last April I told how to convert object code to the Intellec Hex Format (AAL pages 14-18, April, 1984). Both Intel and Zilog use that format. Motorola, on the other hand, has its own format for object code. It is similar, but it is significantly different. If you are programming for a Motorola chip, or using a PROM burner that uses Motorola format, then the following program is for you.

The Motorola S-Type format has three kinds of records: header, data, and end-of-file. Each record begins with the letter "S" and ends with a carriage return linefeed (CRLF). According to the samples I have seen, all of the bytes in a record are in ASCII code with the high bit zero. (Apple peripherals tend to like the high bit = 1, so I made this an option.) The maximum length including the "S" and up to but not including the CRLF is 64 "frames". Between the "S" and CRLF, each record consists of five fields:

> Record format field: ASCII 0, 1, or 9 (hex $30, $31, or $39) for header, data, or end-of-file records respectively.
>
> Byte count field: the count expressed as two ASCII digits of the number of bytes (half the number of frames) from address field through the checksum field. The minimum is 3, and the maximum is 60 decimal or $3C hexadecimal.
>
> Address field: four frames representing the four digits of the load address for data bytes in a data record, or the run address in an end-of-file record. All four digits will be "0" in a header record.
>
> Data field: two hex digits for each byte of data. The number of bytes will be 3 less than the number specified in the byte count field, because that count includes two bytes for the address and one byte for the checksum.
>
> Checksum field: two hex digits representing the 1's complement of the binary sum of all the bytes in the previous four fields.

If you compare the S-Type format with the Intellec format, you will note several differences:

* records start with "S" instead of ":"
* the fields are in a different order
* there was no header record for Intellec
* the byte count covers three fields instead of only the data field
* the checksum is computed by a different algorithm and covers different data.

I tried to use as much as possible of the Intellec program when writing the Motorola program. You will find a lot of similarities if you compare the two. Both are designed to be used with the monitor's control-Y instruction. Both expect you

to enter the output slot number or address in zero-page bytes 0 and 1.

The Motorola program requires two additional pieces of information. It needs a byte at 0002 which will be either $00 or $80, denoting whether to set the high bit to 0 or 1 on every output byte. It also needs an eight character name for the header record. This should be entered in zero-page locations 0003 through 000A.

For example, assume the object code I want to format is in the Apple between $6000 and $67FF. In the target processor it will load at address $1000. The name of the program is "SAMPLE". I want to send the data with the high bit = 0. The device I want to send it to is connected to an intelligent peripheral card in slot 2. Here is what I type:

```
]BRUN B.MOTOROLA FORMATTER       (or :BRUN B.MOTOROLA FORMATTER
]CALL -151                       (or :MNTR)
*0:2 0                           (send to slot 2)
*:0                              (hi-bit = 0)
*:53 41 4D 50 4C 45 20 20        ("SAMPLE")
*1000<6000.67FF^Y                (^Y means control-Y)
```

I recommend comparing this program and my description of it with the Intellec program and article in the April AAL. Here is the Motorola formatter:

```
                *SAVE S.MOTOROLA S-TYPE OBJECT
                       .OR $800
                *-------------------------------
0000-           PORT        .EQ $00,01
0002-           HI.BIT      .EQ $02
0003-           NAME        .EQ $03 ... $0A
0012-           CHECK.SUM   .EQ $12
0013-           TYPE        .EQ $13
0014-           COUNT       .EQ $14
0015-           REMAINING   .EQ $15,16
0017-           START       .EQ $17,18
0019-           END         .EQ $19,1A
001B-           TARGET      .EQ $1B,1C
                *-------------------------------
003C-           A1      .EQ $3C,3D
003E-           A2      .EQ $3E,3F
0040-           A3      .EQ $40,41
0042-           A4      .EQ $42,43
0044-           A5      .EQ $44,45
                *-------------------------------
03F8-           CTRLY.VECTOR          .EQ $3F8 THRU $3FA
03EA-           DOS.REHOOK            .EQ $3EA
                *-------------------------------
FCB4-           MON.NXTA4             .EQ $FCB4
FD8E-           MON.CROUT             .EQ $FD8E
FDDA-           MON.PRHEX             .EQ $FDDA
FDED-           MON.COUT              .EQ $FDED
FE93-           MON.SETVID            .EQ $FE93
                *-------------------------------
                *      SETUP CONTROL-Y
                *-------------------------------
0800- A9 10     SETUP  LDA #SEND.DATA
0802- 8D F9 03         STA CTRLY.VECTOR+1
0805- A9 08            LDA /SEND.DATA
0807- 8D FA 03         STA CTRLY.VECTOR+2
080A- A9 4C            LDA #$4C
080C- 8D F8 03         STA CTRLY.VECTOR
080F- 60               RTS
                *-------------------------------
```

```
                 1390 *     *0:XX YY    (LO,HI OF PORT)
                 1400 *     *:ZZ        (00 OR 80, FOR ASCII HI-BIT)
                 1410 *     *:C1 C2 C3 C4 C5 C6 C7 C8    ASCII VALUES FOR
                 1420 *                 THE 8 CHARACTERS OF THE NAME
                 1430 *     *TARGET<START.END<Y>
                 1440 *        IF PORT IS 0, DO NOT CHANGE OUTPUT
                 1450 *        IF PORT IS 1...7, OUTPUT TO SLOT.
                 1460 *           ELSE OUTPUT TO SUBROUTINE
                 1470 *        SEND BYTES START...END
                 1480 *
                 1490 *        1.  TURN ON OUTPUT PORT
                 1500 *        2.  SEND ID RECORD
                 1510 *        3.  SEND DATA RECORDS
                 1520 *        4.  SEND EOF RECORD
                 1530 *        5.  TURN OFF OUTPUT PORT
                 1540 *--------------------------------
                 1550 SEND.DATA
0810- 20 25 08   1560        JSR SAVE.PARAMETERS
0813- 20 49 08   1570        JSR TURN.ON.OUTPUT.PORT
0816- 20 62 08   1580        JSR SEND.ID.RECORD
0819- 20 37 08   1590        JSR RESTORE.PARAMETERS
081C- 20 7F 08   1600        JSR SEND.DATA.RECORDS
081F- 20 AB 08   1610        JSR SEND.EOF.RECORD
0822- 4C BE 08   1620        JMP TURN.OFF.OUTPUT.PORT
                 1630 *--------------------------------
                 1640 SAVE.PARAMETERS
0825- A2 01      1650        LDX #1
0827- B5 3C      1660 .1     LDA A1,X
0829- 95 17      1670        STA START,X
082B- B5 3E      1680        LDA A2,X
082D- 95 19      1690        STA END,X
082F- B5 42      1700        LDA A4,X
0831- 95 1B      1710        STA TARGET,X
0833- CA         1720        DEX
0834- 10 F1      1730        BPL .1
0836- 60         1740        RTS
                 1750 *--------------------------------
                 1760 RESTORE.PARAMETERS
0837- A2 01      1770        LDX #1
0839- B5 17      1780 .1     LDA START,X
083B- 95 3C      1790        STA A1,X
083D- B5 19      1800        LDA END,X
083F- 95 3E      1810        STA A2,X
0841- B5 1B      1820        LDA TARGET,X
0843- 95 42      1830        STA A4,X
0845- CA         1840        DEX
0846- 10 F1      1850        BPL .1
0848- 60         1860        RTS
                 1870 *--------------------------------
                 1880 TURN.ON.OUTPUT.PORT
0849- A6 01      1890        LDX PORT+1     HI-BYTE OF PORT SPECIFIED
084B- D0 0A      1900        BNE .1
084D- A5 00      1910        LDA PORT       LO-BYTE, MUST BE SLOT
084F- 29 07      1920        AND #$07
0851- F0 0E      1930        BEQ .3       SLOT 0, JUST SCREEN
0853- 09 C0      1940        ORA #$C0
0855- D0 03      1950        BNE .2       ...ALWAYS
0857- 8A         1960 .1     TXA          HI-BYTE OF SUBROUTINE
0858- A6 00      1970        LDX PORT       LO-BYTE OF SUBROUTINE
085A- 85 37      1980 .2     STA $37
085C- 86 36      1990        STX $36
085E- 20 EA 03   2000        JSR DOS.REHOOK
0861- 60         2010 .3     RTS
                 2020 *--------------------------------
                 2030 SEND.ID.RECORD
0862- A9 30      2040        LDA #'0'     TYPE = "0"
0864- 85 13      2050        STA TYPE
0866- A9 08      2060        LDA #8       COUNT = 8
0868- 85 14      2070        STA COUNT
086A- A9 00      2080        LDA #0       ADDR=0
086C- 85 42      2090        STA A4
086E- 85 43      2100        STA A4+1
0870- 85 3D      2110        STA A1+1
0872- 85 3F      2120        STA A2+1
```

```
0874- A9 03    2130        LDA #NAME
0876- 85 3C    2140        STA A1
0878- A9 0A    2150        LDA #NAME+7
087A- 85 3E    2160        STA A2
087C- 4C C4 08 2170        JMP SEND.RECORD
               2180 *--------------------------------
               2190 SEND.DATA.RECORDS
087F- A9 31    2200        LDA #'1'     TYPE = "1"
0881- 85 13    2210        STA TYPE
0883- E6 3E    2220        INC A2       POINT JUST BEYOND THE END
0885- D0 02    2230        BNE .1
0887- E6 3F    2240        INC A2+1
0889- 38       2250 .1     SEC
088A- A2 14    2260        LDX #20
088C- A5 3E    2270        LDA A2       SEE HOW MANY BYTES LEFT
088E- E5 3C    2280        SBC A1
0890- 85 15    2290        STA REMAINING
0892- A5 3F    2300        LDA A2+1
0894- E5 3D    2310        SBC A1+1
0896- 85 16    2320        STA REMAINING+1
0898- D0 08    2330        BNE .2       USE MIN(20,A2-A1+1)
089A- E4 15    2340        CPX REMAINING
089C- 90 04    2350        BCC .2
089E- A6 15    2360        LDX REMAINING
08A0- F0 08    2370        BEQ .3       ...FINISHED
08A2- 86 14    2380 .2     STX COUNT
08A4- 20 C4 08 2390        JSR SEND.RECORD
08A7- 4C 89 08 2400        JMP .1       ...ALWAYS
08AA- 60       2410 .3     RTS
               2420 *--------------------------------
               2430 SEND.EOF.RECORD
08AB- A9 00    2440        LDA #0       # OF DATA BYTES = 0
08AD- 85 14    2450        STA COUNT
08AF- A9 39    2460        LDA #'9'     TYPE="9"
08B1- 85 13    2470        STA TYPE
08B3- A5 1B    2480        LDA TARGET   RUN ADDRESS (LO)
08B5- 85 42    2490        STA A4
08B7- A5 1C    2500        LDA TARGET+1 RUN ADDRESS (HI)
08B9- 85 43    2510        STA A4+1
08BB- 4C C4 08 2520        JMP SEND.RECORD
               2530 *--------------------------------
               2540 TURN.OFF.OUTPUT.PORT
08BE- 20 93 FE 2550        JSR MON.SETVID
08C1- 4C EA 03 2560        JMP DOS.REHOOK
               2570 *--------------------------------
               2580 SEND.RECORD
08C4- A9 53    2590        LDA #'S'     LETTER "S"
08C6- 20 21 09 2600        JSR SEND.FRAME
08C9- A5 13    2610        LDA TYPE     TYPE "0", "1", OR "9"
08CB- 20 21 09 2620        JSR SEND.FRAME
08CE- A9 00    2630        LDA #0       INIT CHECKSUM
08D0- 85 12    2640        STA CHECK.SUM
08D2- 18       2650        CLC
08D3- A5 14    2660        LDA COUNT    SEND BYTE COUNT
08D5- 69 03    2670        ADC #3       ...INCLUDING ADDR AND CSUM
08D7- 20 07 09 2680        JSR SEND.BYTE
08DA- A5 43    2690        LDA A4+1     SEND ADDRESS
08DC- 20 07 09 2700        JSR SEND.BYTE
08DF- A5 42    2710        LDA A4
08E1- 20 07 09 2720        JSR SEND.BYTE
08E4- A5 14    2730        LDA COUNT    ANY DATA?
08E6- F0 0E    2740        BEQ .2       ...NO
08E8- A0 00    2750        LDY #0       ...YES, SEND DATA
08EA- B1 3C    2760 .1     LDA (A1),Y
08EC- 20 07 09 2770        JSR SEND.BYTE
08EF- 20 B4 FC 2780        JSR MON.NXTA4
08F2- C6 14    2790        DEC COUNT
08F4- D0 F4    2800        BNE .1
08F6- A5 12    2810 .2     LDA CHECK.SUM     SEND CHECK SUM
08F8- 49 FF    2820        EOR #$FF
08FA- 20 07 09 2830        JSR SEND.BYTE
08FD- A9 0D    2840        LDA #$0D     SEND CRLF
08FF- 20 21 09 2850        JSR SEND.FRAME
0902- A9 0A    2860        LDA #$0A     LINEFEED
0904- 4C 21 09 2870        JMP SEND.FRAME
```

```
                  *--------------------------------
                  SEND.BYTE
0907- 48                  PHA           SAVE BYTE
0908- 18                  CLC
0909- 65 12               ADC CHECK.SUM     ACCUMULATE CHECKSUM
090B- 85 12               STA CHECK.SUM
090D- 68                  PLA           RECOVER BYTE
090E- 48                  PHA           SAVE ANOTHER COPY
090F- 4A                  LSR           READY FIRST DIGIT
0910- 4A                  LSR
0911- 4A                  LSR
0912- 4A                  LSR
0913- 20 19 09            JSR SEND.DIGIT
0916- 68                  PLA           RECOVER BYTE
0917- 29 0F               AND #$0F      READY SECOND DIGIT
                  SEND.DIGIT
0919- 09 30               ORA #$30      CHANGE TO ASCII
091B- C9 3A               CMP #$3A
091D- 90 02               BCC SEND.FRAME
091F- 69 06               ADC #6        CHANGE TO A...F
                  SEND.FRAME
0921- 05 02               ORA HI.BIT    $00 OR $80
0923- 4C ED FD            JMP MON.COUT
                  *--------------------------------
                          .OR $300
                  SAMPLE
0300- 86 44 B7
0303- 01 00 41
0306- 42 43               .HS 86.44.B7.01.00.41.42.43
0308- 44 45 46
030B- 47 48 49
030E- 4A 4B               .HS 44.45.46.47.48.49.4A.4B
0310- 4C 4D 4E            .HS 4C.4D.4E
```

Making a 65C02 Work in my Apple II Plus........William O'Ryan

I am writing this on my Apple II Plus running a 2 MHz 65C02 (GTE G65SC02PI-2). All is well now, but it took some doing.

A few days after pluggin in the chip I started noticing problems. Applesoft found itself unable to process numeric literals, and the version of FORTH I have been developing began acting weird.

Following the tip in AAL that the timing of the fetch-process-save instructions might be responsible, I ran some tests on them. The 65C02 worked flawlessly. Apparently the problem is elsewhere.

After further checking, especially in my FORTH, I found that a certain BNE instruction sitting in the first byte of a page and branching backward into the prior page frequently branched back one byte less than it should.

I'm not a hardware person, but I figured debugging is debugging and I really wanted that chip to work, so I began staring at the circuit diagram in the Apple Reference manual. After several hours I concluded that I stood for input, O for output, D for data, and A for address.

The easiest hypothesis to check seemed to be that data was not getting back from the RAMs to the microprocessor in time. So I wrote down some chip numbers and went downtown to see if I could buy some faster varients. Well, the first two chips I replaced solved the problem.

They were 74LS257 chips at B6 and B7. These chips multiplex the output of RAM with the output of the keyboard and send the result to the 65C02. I replaced them with 74F257 chips. I understand these consume less power, respond faster, and are more susceptible to electrostatic damage.

Anyway, my 65C02 is happy now. I would like to hear whether this modification works in other Apples, and with other 65C02s. Drop a line to Bob and Bill at S-C if you have any word on this.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $12 postage for other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).